F1 Database Report

Created by: Shuber Ali Mirza

ID: 20027047

Database Systems – ISYS2014

ASSIGNMENT

Contents

Introduction: 2
Design: 2
  Choosing the Entities: 2
  ER Diagram: 3
  Cardinality: 4
  Participation: 4
  Relationship schema mapping (Using the slides): 5
  Normalization: 5
  Final Schema Mapping: 5
Datatypes and Descriptions: 6
  Tyres: 6
  Teams: 6
  Drivers: 6
  Winners: 6
  Suppliers: 6
  Circuits: 7
  Races: 7
Implementation: 7
Design and Use: 8
  Query Questions, Implementations, and Outputs: 8
  Advanced Features: 12
    Procedures: 12
    Triggers: 15
    Views: 15
  Python Connectivity: 16
Final Discussion: 16
  Challenges: 16
  Future Improvements: 16

Introduction:

The formula 1 motor racing sport is a global phenomenon, that is takes place every, over a season of around 8 or 9 months. The season has a variable number races each year, called grand prix, having 22 races in the 2022 season, that take place in different countries around the globe. There are 10 teams that are allowed to compete at this top level, and each can appoint two drivers to represent them during races. The whole point of the sport is for the teams to engineer the fastest and most reliable cars, and appoint competent drivers to drive them, to win as many races as possible.

This report will thoroughly go through the design, implementation, and testing of a database created to get basic information related to, and follow along a fictional season of formula 1.

Design:

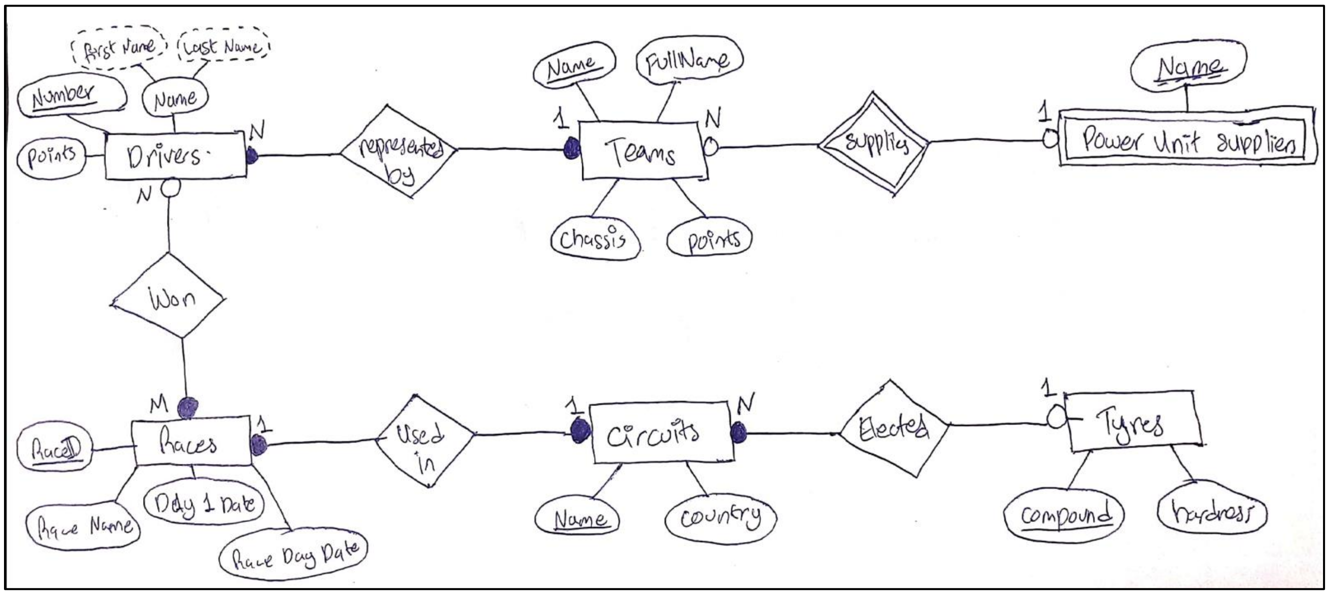
Choosing the Entities:

When thinking about the requirements of the assignment, and the freedom of choice it has given, selecting the entities was quite simple, as I could choose the things that I was personally most interested in. These are all chosen so that they would be complex enough to display an affinity in MySQL and still keeping them simple enough for any user to understand.

1. Teams: These are the building blocks of formula 1, thus having an entity relating to that is a no brainer. Basic information about the team, found on the official website, will be inserted here.
2. Power Unit Suppliers: This entity was chosen to demonstrate the use of weak entities in the design stage, and how composite keys can be used to resolve their case. All the suppliers that supply the power units (engines) for the cars used by the teams, are listed, with the team they are supplier to being required to fulfill the weak entity constraint.
3. Drivers: This entity will be used insert basic information about the drivers, relative to the current point in the season. These come hand-in-hand with the teams, entity, so a relationship connecting them is made.

4. Tyres: This entity is used to store information about the 5 different compounds of tyres used for the season. It simply tells what ID of tyre compound relates to what type of hardness the tyre has.
5. Circuits: This entity will be used to add basic information about the circuits that will be used during the grand prix. They include information about the where they are located and what is the hardest compound of tyre used for racing. Only the hardest compound is used as an attribute, as it can be used to derive the other two compounds that are used, utilizing a relationship with the Tyres entity, reducing redundancy in the entity.
6. Races: Used to store information such as ID of the race, circuit the race is held on and when it starts or started. Since circuits are used in races, a relationship is made.
7. Winners: Each race is going to have a winning driver, thus creating a relationship between those entities. This is used to keep track of which driver has won which race.

ER Diagram:



Cardinality:

Relationship	Entities	Cardinality
Represented By	Teams, Drivers	1:N Each team can be represented by many drivers Each driver represents one team
Won	Drivers, Races	N:M Drivers can win many races Races won by many drivers
Used In	Circuits, Races	1:1 Each circuit used for only one race Each race uses only one circuit
Elected	Circuits, Tyres	1:N Each tyre can be elected for many circuits Each circuit elects only one tyre (Due to there only being one attribute for tyre compound in circuits entity)
Supplies	Suppliers, Teams	1:N Each supplier can supply to many teams Each team is supplied by one supplier

Participation:

Relationship	Entities	Participation
Represented By	Teams, Drivers	All teams must be represented by drivers (Full) All drivers must represent a team (Full)
Won	Drivers, Races	All races must have a winner (Full) Not all drivers have won a race (Partial)
Used In	Circuits, Races	All circuits (in the database) must be used in a races (Full) All races must use a circuit (Full)
Elected	Circuits, Tyres	Not all tyres are elected in circuits (since only the hardest compound is stored in Circuits entity) (Partial) All circuits must elect a tyre (Full)
Supplies	Suppliers, Teams	All suppliers (in the database) must supply to a team (Full) Not all teams are supplied by a supplier (Partial)

Relationship schema mapping (Using the slides):

1. Circuits(Circuit Name, Country, Hardest Tyre Compound, Race ID, Race Name, Day 1 Date, Race Day Date)
2. Tyres (Compound, Hardness)
3. Teams (Name, Full Name, Power Unit, Chassis, Current Points)
4. Drivers (Number, First Name, Last Name, Team Name, Current Points)
5. Winners (Race ID, Driver Number)
6. Power Unit Suppliers (Name, Team name)

Normalization:

All in 1nf, with no multi-valued attributes.

Circuits relation not in 2nf, due to partial functional dependencies:

- Circuit Name -> Country, Hardest Tyre Compound
- Race ID -> Race Name, Day 1 Date, Race Day Date

Therefore, Decomposed into two relations:

1. Circuits (Name, Country, Hardest tyre compound)
2. Races (Race ID, Race Name, Circuit Name, Day 1 Date, Race Day date)

Finally, all relations in 3nf, with no attributes dependent on non-prime attributes in any of them.

Final Schema Mapping:

1. Tyres (Compound, Hardness)
2. Teams (Name, Full Name, Power Unit, Chassis, Current Points)
3. Drivers (Number, First Name, Last Name, Team Name, Current Points)
4. Winners (Race ID, Driver Number)
5. Power Unit Suppliers (Name, Team name)
6. Circuits (Name, Country, Hardest tyre compound)
7. Races (Race ID, Race Name, Circuit Name, Day 1 Date, Race Day date)

Datatypes and Descriptions:

Tyres:

Attribute	Datatype	Size	NULL / Not NULL	Description
compound	CHAR	2	Not NULL	Name of compound of tyre, primary key
hardness	VARCHAR	20	Not NULL	How hard or soft the compound is

Teams:

Attribute	Datatype	Size	NULL / Not NULL	Description
teamName	VARCHAR	15	Not NULL	Name of team, primary key
fullName	VARCHAR	40	Not NULL	Full name of the team
pUnitSupplier	VARCHAR	30	NULL	Name of the power unit supplier, foreign key
chassis	CHAR	7	Not NULL	Name of the chassis the team cars use
points	DECIMAL	5,1	NULL	Points the team currently has

Drivers:

Attribute	Datatype	Size	NULL / Not NULL	Description
dNo	CHAR	3	Not NULL	Driver's Number, primary key
fName	VARCHAR	10	Not NULL	First name of driver
lName	VARCHAR	10	NULL	Last name of driver
teamName	VARCHAR	15	Not NULL	Team the driver drives for, foreign key
points	DECIMAL	5,1	NULL	Points the driver currently has

Winners:

Attribute	Datatype	Size	NULL / Not NULL	Description
raceID	INT		Not NULL	ID of race, foreign key
dNo	CHAR	3	Not NULL	Driver's Number, foreign key

Suppliers:

Attribute	Datatype	Size	NULL / Not NULL	Description
name	VARCHAR	30	Not NULL	Name of power unit supplier, composite key
teamName	VARCHAR	15	Not NULL	Name of team it supplies to, composite/foreign key

Circuits:

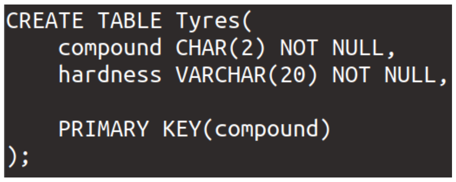
Attribute	Datatype	Size	NULL / Not NULL	Description
circuitName	VARCHAR	30	Not NULL	Name of circuit, primary key
country	VARCHAR	15	Not NULL	Country circuit is in
hardestTyre	CHAR	2	Not NULL	Hardest tyre compound used in the circuit, foreign key

Races:

Attribute	Datatype	Size	NULL / Not NULL	Description
raceID	INT		Not NULL	ID of race, primary key
raceName	VARCHAR	70	Not NULL	Name of race
circuitName	VARCHAR	30	Not NULL	Name of circuit, foreign key
day1Date	DATE		Not NULL	Date of first day of race
raceDate	DATE		Not NULL	Date of grand prix

Implementation:

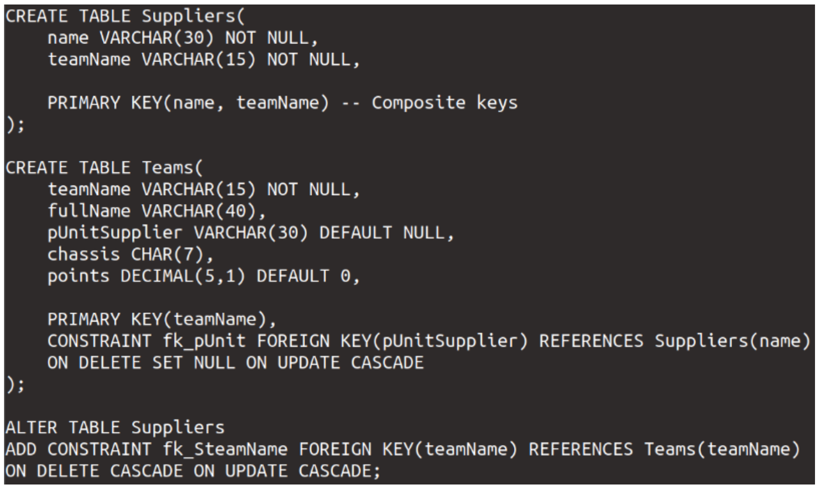
Using the design scheme discussed, the database was created.



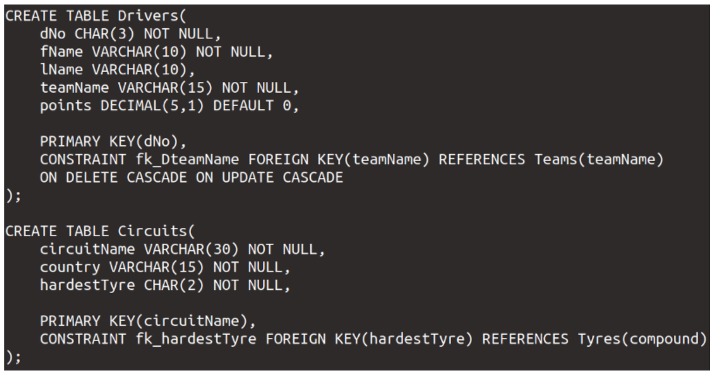
```
CREATE TABLE Tyres(
    compound CHAR(2) NOT NULL,
    hardness VARCHAR(20) NOT NULL,

    PRIMARY KEY(compound)
);
```

The Tyres table, created with the designed datatypes and constraints

The Suppliers and Teams tables were created as designed, with name and teamName as composite keys in the Suppliers table, as it is a weak entity, that requires teamName to define a record. The table is altered after the creation of the Teams table, to add teamName as a foreign key. On delete and on update constraints for foreign keys as appropriate.

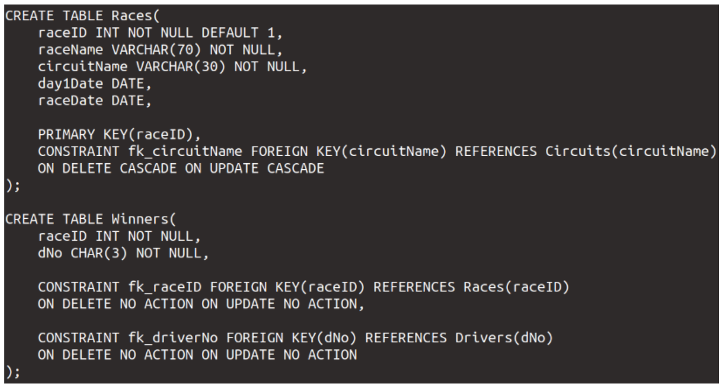


```
CREATE TABLE Suppliers(
    name VARCHAR(30) NOT NULL,
    teamName VARCHAR(15) NOT NULL,

    PRIMARY KEY(name, teamName) -- Composite keys
);

CREATE TABLE Teams(
    teamName VARCHAR(15) NOT NULL,
    fullName VARCHAR(40),
    pUnitSupplier VARCHAR(30) DEFAULT NULL,
    chassis CHAR(7),
    points DECIMAL(5,1) DEFAULT 0,

    PRIMARY KEY(teamName),
    CONSTRAINT fk_pUnit FOREIGN KEY(pUnitSupplier) REFERENCES Suppliers(name)
    ON DELETE SET NULL ON UPDATE CASCADE
);

ALTER TABLE Suppliers
ADD CONSTRAINT fk_SteamName FOREIGN KEY(teamName) REFERENCES Teams(teamName)
ON DELETE CASCADE ON UPDATE CASCADE;
```



```
CREATE TABLE Drivers(
    dNo CHAR(3) NOT NULL,
    fName VARCHAR(10) NOT NULL,
    lName VARCHAR(10),
    teamName VARCHAR(15) NOT NULL,
    points DECIMAL(5,1) DEFAULT 0,

    PRIMARY KEY(dNo),
    CONSTRAINT fk_DteamName FOREIGN KEY(teamName) REFERENCES Teams(teamName)
    ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE Circuits(
    circuitName VARCHAR(30) NOT NULL,
    country VARCHAR(15) NOT NULL,
    hardestTyre CHAR(2) NOT NULL,

    PRIMARY KEY(circuitName),
    CONSTRAINT fk_hardestTyre FOREIGN KEY(hardestTyre) REFERENCES Tyres(compound)
);
```

The Drivers and Circuits tables, created as designed, with appropriate on delete and on update foreign key constraints added.



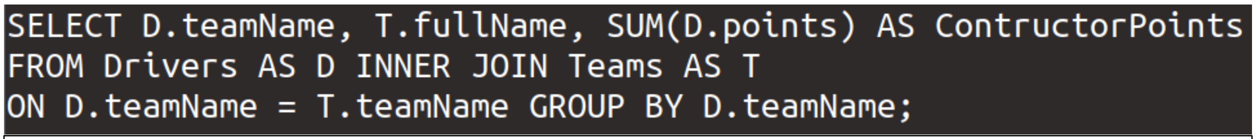
```
CREATE TABLE Races(
    raceID INT NOT NULL DEFAULT 1,
    raceName VARCHAR(70) NOT NULL,
    circuitName VARCHAR(30) NOT NULL,
    day1Date DATE,
    raceDate DATE,

    PRIMARY KEY(raceID),
    CONSTRAINT fk_circuitName FOREIGN KEY(circuitName) REFERENCES Circuits(circuitName)
    ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE Winners(
    raceID INT NOT NULL,
    dNo CHAR(3) NOT NULL,

    CONSTRAINT fk_raceID FOREIGN KEY(raceID) REFERENCES Races(raceID)
    ON DELETE NO ACTION ON UPDATE NO ACTION,

    CONSTRAINT fk_driverNo FOREIGN KEY(dNo) REFERENCES Drivers(dNo)
    ON DELETE NO ACTION ON UPDATE NO ACTION
);
```

The Races and Winners tables, created as designed, each with appropriate on delete and on update foreign key constraints added. Winners table has foreign keys from two tables, Races and Drivers.

Design and Use:

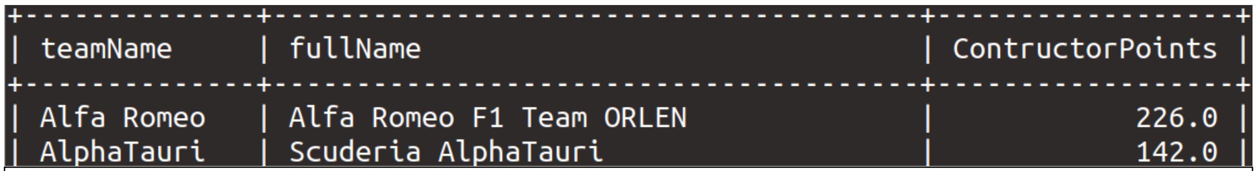
Query Questions, Implementations, and Outputs:

1. What are the current points of the constructors, depending on their driver's points?



```
SELECT D.teamName, T.fullName, SUM(D.points) AS ContructorPoints
FROM Drivers AS D INNER JOIN Teams AS T
ON D.teamName = T.teamName GROUP BY D.teamName;
```

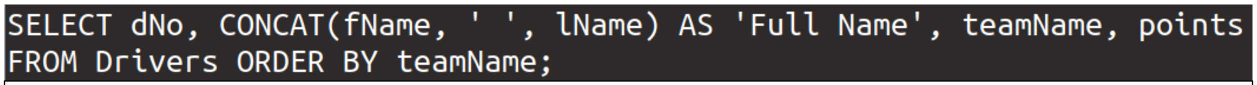
Query to show the current points that teams have. Using the SUM() aggregate function and GROUP BY, adds the points of drivers representing the teams.



```
+-------------+----------------------------------------+------------------+
| teamName    | fullName                               | ContructorPoints |
+-------------+----------------------------------------+------------------+
| Alfa Romeo  | Alfa Romeo F1 Team ORLEN               |            226.0 |
| AlphaTauri  | Scuderia AlphaTauri                    |            142.0 |
```

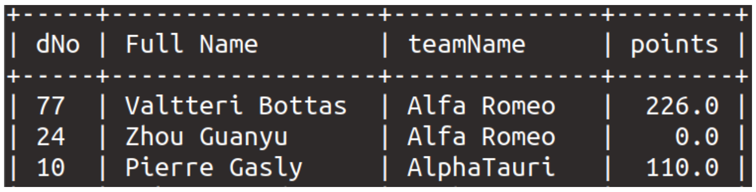
Output from query. Only first two tuples displayed

2. What are the current points of drivers, with their full name, and teams they belong to, ordered by teams?



```
SELECT dNo, CONCAT(fName, ' ', lName) AS 'Full Name', teamName, points
FROM Drivers ORDER BY teamName;
```

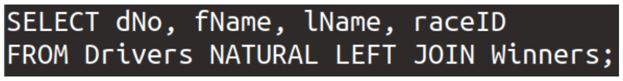
Query for current driver points. Use of concatenation function CONCAT() to derive the full names of the drivers, and use of ORDER BY.



```
+-----+------------------+-------------+--------+
| dNo | Full Name        | teamName    | points |
+-----+------------------+-------------+--------+
| 77  | Valtteri Bottas  | Alfa Romeo  |  226.0 |
| 24  | Zhou Guanyu      | Alfa Romeo  |    0.0 |
| 10  | Pierre Gasly     | AlphaTauri  |  110.0 |
```

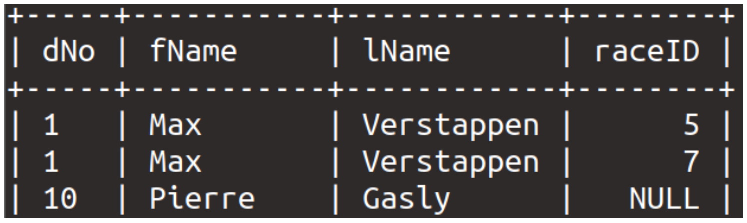
Sample output from the query. Only first 3 tuples displayed, but shows ordered by teams

3. What are the current points of drivers, and have they won any races?



```
SELECT dNo, fName, lName, raceID
FROM Drivers NATURAL LEFT JOIN Winners;
```

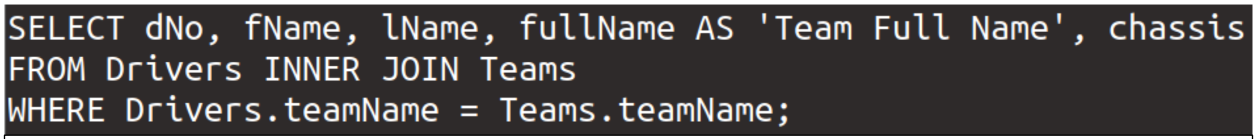
Query for current points, with wins



```
+-----+-----------+------------+--------+
| dNo | fName     | lName      | raceID |
+-----+-----------+------------+--------+
| 1   | Max       | Verstappen |      5 |
| 1   | Max       | Verstappen |      7 |
| 10  | Pierre    | Gasly      |   NULL |
```

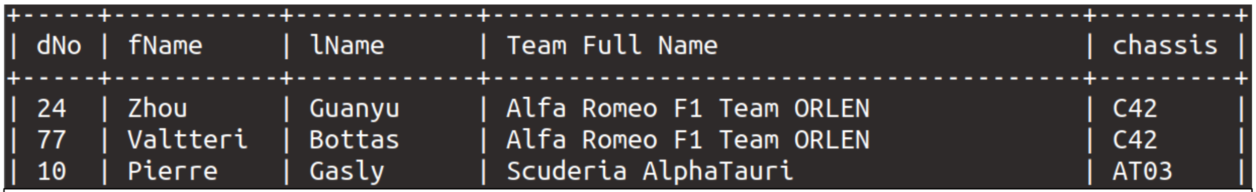
Sample output with only 3 tuples displayed. NULL value, due to NATURAL JOIN, in the raceID attribute shows wins

4. What teams do drivers belong to, with full names of the teams and the car chassis they drive?



```
SELECT dNo, fName, lName, fullName AS 'Team Full Name', chassis
FROM Drivers INNER JOIN Teams
WHERE Drivers.teamName = Teams.teamName;
```

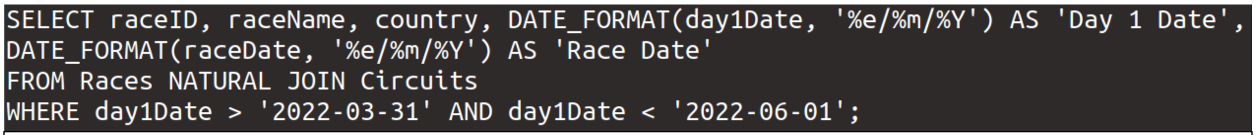
Query for drivers and teams, with full names of the teams got from INNER JOIN



```
+-----+-----------+------------+--------------------------------------+---------+
| dNo | fName     | lName      | Team Full Name                       | chassis |
+-----+-----------+------------+--------------------------------------+---------+
| 24  | Zhou      | Guanyu     | Alfa Romeo F1 Team ORLEN             | C42     |
| 77  | Valtteri  | Bottas     | Alfa Romeo F1 Team ORLEN             | C42     |
| 10  | Pierre    | Gasly      | Scuderia AlphaTauri                  | AT03    |
```

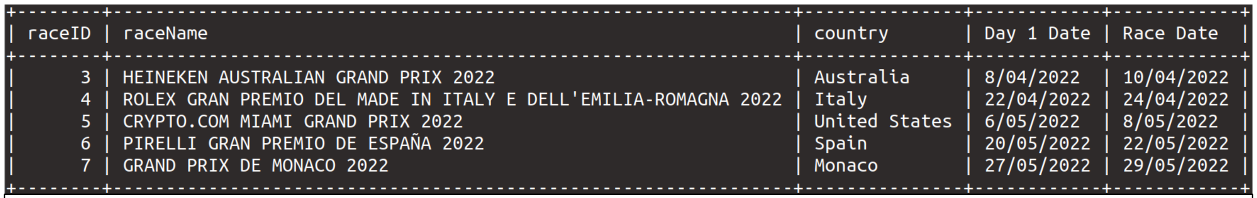
Sample output from the query. Only first three tuples displayed, showing the INNER JOIN working, with team's full name and car chassis displayed

5. What races take place after March, but before June, and where do they take place, shown in local date format (DD/MM/YYYY)?



```
SELECT raceID, raceName, country, DATE_FORMAT(day1Date, '%e/%m/%Y') AS 'Day 1 Date',
DATE_FORMAT(raceDate, '%e/%m/%Y') AS 'Race Date'
FROM Races NATURAL JOIN Circuits
WHERE day1Date > '2022-03-31' AND day1Date < '2022-06-01';
```

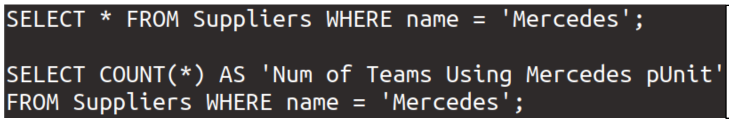
Query for finding races in between the dates. Use of DATE_FORMAT() function to show dates in the local format, and NATURAL JOIN to find the countries



```
+--------+-------------------------------------------------------------------+---------------+------------+------------+
| raceID | raceName                                                          | country       | Day 1 Date | Race Date  |
+--------+-------------------------------------------------------------------+---------------+------------+------------+
|      3 | HEINEKEN AUSTRALIAN GRAND PRIX 2022                               | Australia     | 8/04/2022  | 10/04/2022 |
|      4 | ROLEX GRAN PREMIO DEL MADE IN ITALY E DELL'EMILIA-ROMAGNA 2022    | Italy         | 22/04/2022 | 24/04/2022 |
|      5 | CRYPTO.COM MIAMI GRAND PRIX 2022                                  | United States | 6/05/2022  | 8/05/2022  |
|      6 | PIRELLI GRAN PREMIO DE ESPAÑA 2022                                | Spain         | 20/05/2022 | 22/05/2022 |
|      7 | GRAND PRIX DE MONACO 2022                                         | Monaco        | 27/05/2022 | 29/05/2022 |
+--------+-------------------------------------------------------------------+---------------+------------+------------+
```

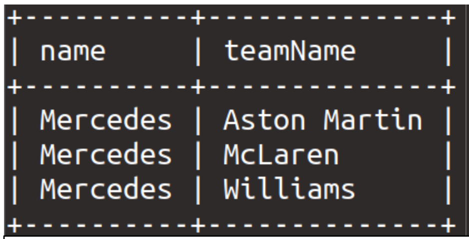
Sample output from the query. Whole output shown, demonstrating the range specified works

6. How many power units does Mercedes supply, excluding themselves, and what are the teams that use them?

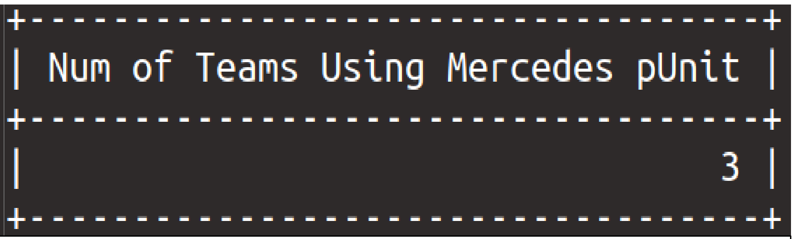


```
SELECT * FROM Suppliers WHERE name = 'Mercedes';

SELECT COUNT(*) AS 'Num of Teams Using Mercedes pUnit'
FROM Suppliers WHERE name = 'Mercedes';
```

Queries to find who Mercedes supplies, and how many teams



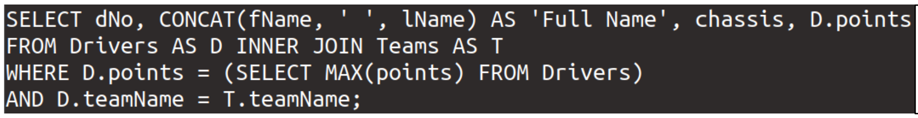
```
+----------+--------------+
| name     | teamName     |
+----------+--------------+
| Mercedes | Aston Martin |
| Mercedes | McLaren      |
| Mercedes | Williams     |
+----------+--------------+
```



```
+-----------------------------------+
| Num of Teams Using Mercedes pUnit |
+-----------------------------------+
|                                 3 |
+-----------------------------------+
```

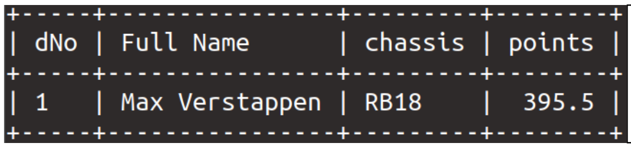
Two sample outputs from the two queries. Use of the COUNT() aggregate function to find number of units supplied.

7. Who is the leader in points for the drivers, with their full name and chassis name?



```
SELECT dNo, CONCAT(fName, ' ', lName) AS 'Full Name', chassis, D.points
FROM Drivers AS D INNER JOIN Teams AS T
WHERE D.points = (SELECT MAX(points) FROM Drivers)
AND D.teamName = T.teamName;
```

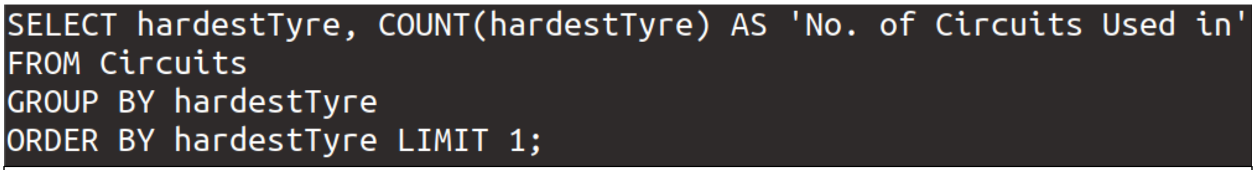
Query to show current leader



```
+-----+----------------+---------+--------+
| dNo | Full Name      | chassis | points |
+-----+----------------+---------+--------+
| 1   | Max Verstappen | RB18    |  395.5 |
+-----+----------------+---------+--------+
```

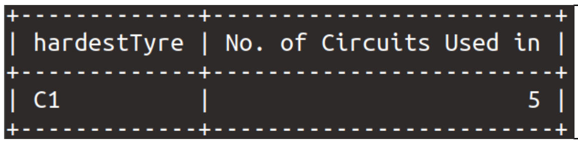
Sample output, showing current highest-ranking driver in the standings

8. What is the most used hard tyre compound this year?



```
SELECT hardestTyre, COUNT(hardestTyre) AS 'No. of Circuits Used in'
FROM Circuits
GROUP BY hardestTyre
ORDER BY hardestTyre LIMIT 1;
```

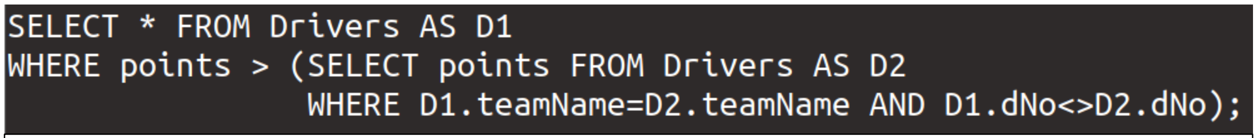
Query showing the current most used hardest compound tyre. Use of COUNT() aggregate function with GROUP BY, and ORDER BY with LIMIT, to show only top



```
+-------------+-------------------------+
| hardestTyre | No. of Circuits Used in |
+-------------+-------------------------+
| C1          |                       5 |
+-------------+-------------------------+
```

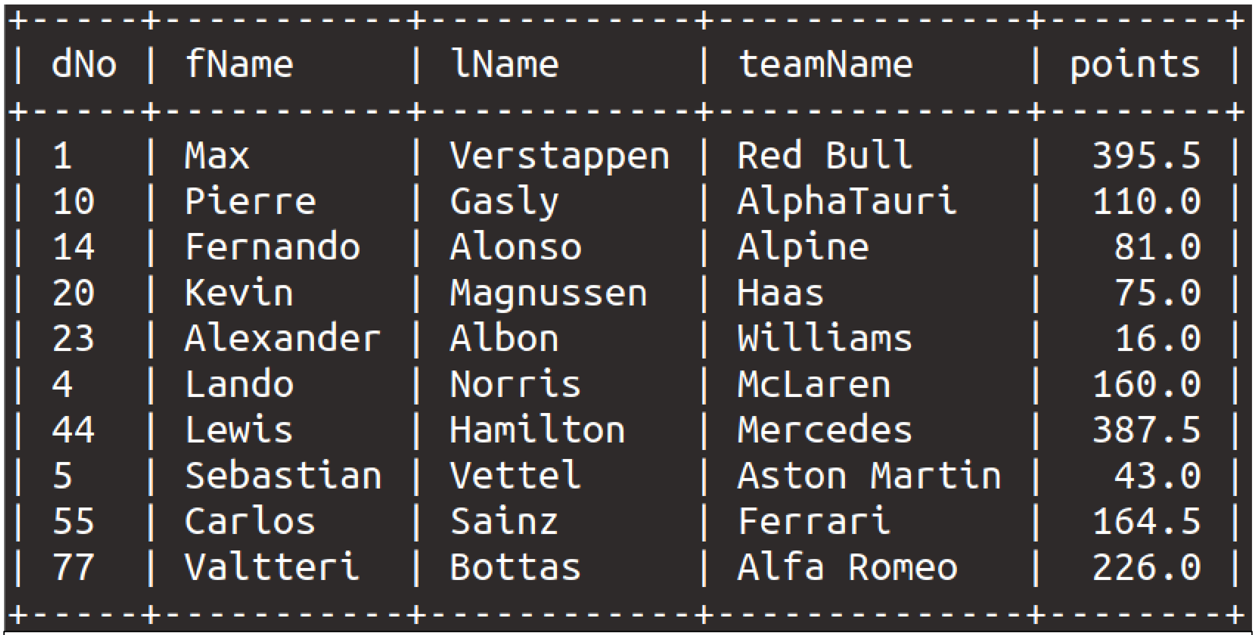
Sample output for the query, showing most repeated tyre, with count of repeats

9. What drivers have higher points, belonging to the same team?



```
SELECT * FROM Drivers AS D1
WHERE points > (SELECT points FROM Drivers AS D2
                WHERE D1.teamName=D2.teamName AND D1.dNo<>D2.dNo);
```

Query to find one driver from each team, with the higher number of points compared to the other. Implemented using subqueries, with the same table



```
+-----+-----------+------------+--------------+--------+
| dNo | fName     | lName      | teamName     | points |
+-----+-----------+------------+--------------+--------+
| 1   | Max       | Verstappen | Red Bull     |  395.5 |
| 10  | Pierre    | Gasly      | AlphaTauri   |  110.0 |
| 14  | Fernando  | Alonso     | Alpine       |   81.0 |
| 20  | Kevin     | Magnussen  | Haas         |   75.0 |
| 23  | Alexander | Albon      | Williams     |   16.0 |
| 4   | Lando     | Norris     | McLaren      |  160.0 |
| 44  | Lewis     | Hamilton   | Mercedes     |  387.5 |
| 5   | Sebastian | Vettel     | Aston Martin |   43.0 |
| 55  | Carlos    | Sainz      | Ferrari      |  164.5 |
| 77  | Valtteri  | Bottas     | Alfa Romeo   |  226.0 |
+-----+-----------+------------+--------------+--------+
```

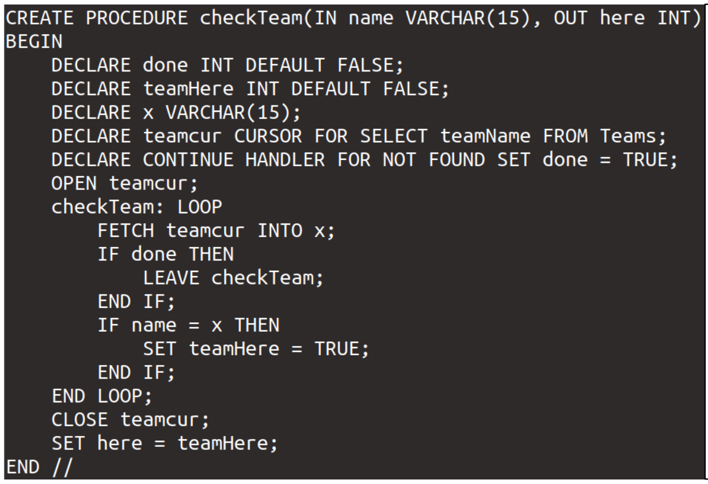
Whole sample output displayed. No repeated team name, showing only one driver from each team displayed, only the one with higher points.

Advanced Features:

Three types of advanced features learnt through the course of this unit have been implemented in this Formula 1 database.

Procedures:

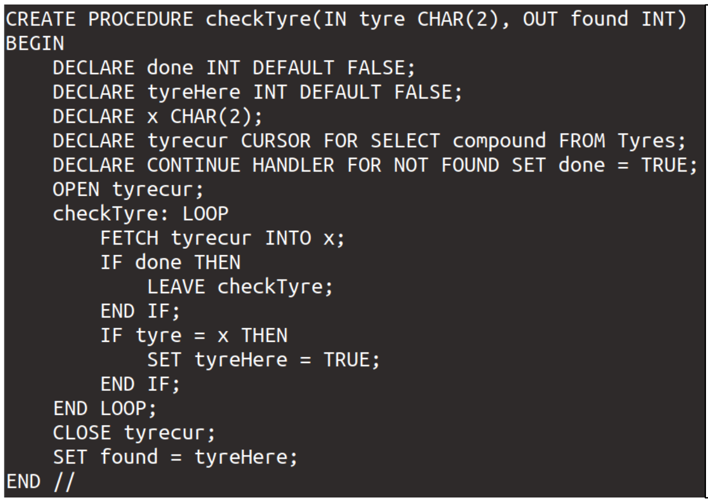
1. checkTeam()



```
CREATE PROCEDURE checkTeam(IN name VARCHAR(15), OUT here INT)
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE teamHere INT DEFAULT FALSE;
    DECLARE x VARCHAR(15);
    DECLARE teamcur CURSOR FOR SELECT teamName FROM Teams;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
    OPEN teamcur;
    checkTeam: LOOP
        FETCH teamcur INTO x;
        IF done THEN
            LEAVE checkTeam;
        END IF;
        IF name = x THEN
            SET teamHere = TRUE;
        END IF;
    END LOOP;
    CLOSE teamcur;
    SET here = teamHere;
END //
```

Procedure used to check if teamName foreign key exists or not. Used in other procedures as a check. Can be used by users to check if a team exists in the Teams table by entering the team’s name as an IN parameter and give an INT variable for the OUT parameter. Returns; TRUE if exists, FALSE if not exists.

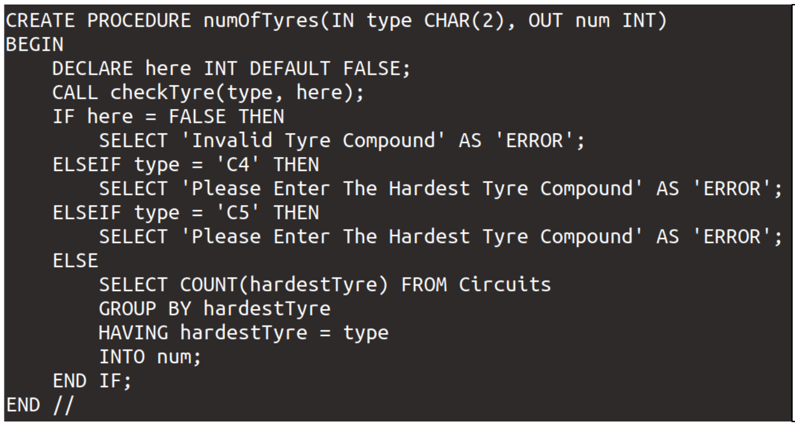
2. checkTyre()



```
CREATE PROCEDURE checkTyre(IN tyre CHAR(2), OUT found INT)
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE tyreHere INT DEFAULT FALSE;
    DECLARE x CHAR(2);
    DECLARE tyrecur CURSOR FOR SELECT compound FROM Tyres;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
    OPEN tyrecur;
    checkTyre: LOOP
        FETCH tyrecur INTO x;
        IF done THEN
            LEAVE checkTyre;
        END IF;
        IF tyre = x THEN
            SET tyreHere = TRUE;
        END IF;
    END LOOP;
    CLOSE tyrecur;
    SET found = tyreHere;
END //
```

Procedure used to check if compound foreign key exists in the Tyres table or not. Used in other procedures as a check. Can be used by users to check if a tyre compound exists in the table by giving a CHAR(2) variable as the IN parameter, and an INT variable as the OUT. Returns; TRUE if exists, FALSE if not exists.

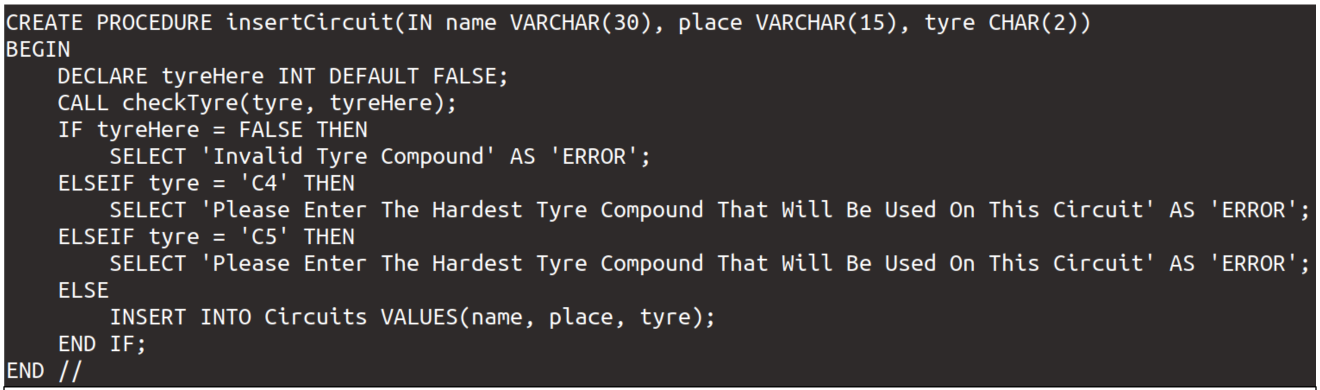
3. numOfTyres()



```
CREATE PROCEDURE numOfTyres(IN type CHAR(2), OUT num INT)
BEGIN
    DECLARE here INT DEFAULT FALSE;
    CALL checkTyre(type, here);
    IF here = FALSE THEN
        SELECT 'Invalid Tyre Compound' AS 'ERROR';
    ELSEIF type = 'C4' THEN
        SELECT 'Please Enter The Hardest Tyre Compound' AS 'ERROR';
    ELSEIF type = 'C5' THEN
        SELECT 'Please Enter The Hardest Tyre Compound' AS 'ERROR';
    ELSE
        SELECT COUNT(hardestTyre) FROM Circuits
        GROUP BY hardestTyre
        HAVING hardestTyre = type
        INTO num;
    END IF;
END //
```

Procedure used to find the number of times a tyre compound has been used in circuits. Uses the checkTyre() stored procedure to check the foreign key constraint. Returns the No. of repeats as an INT using the OUT parameter

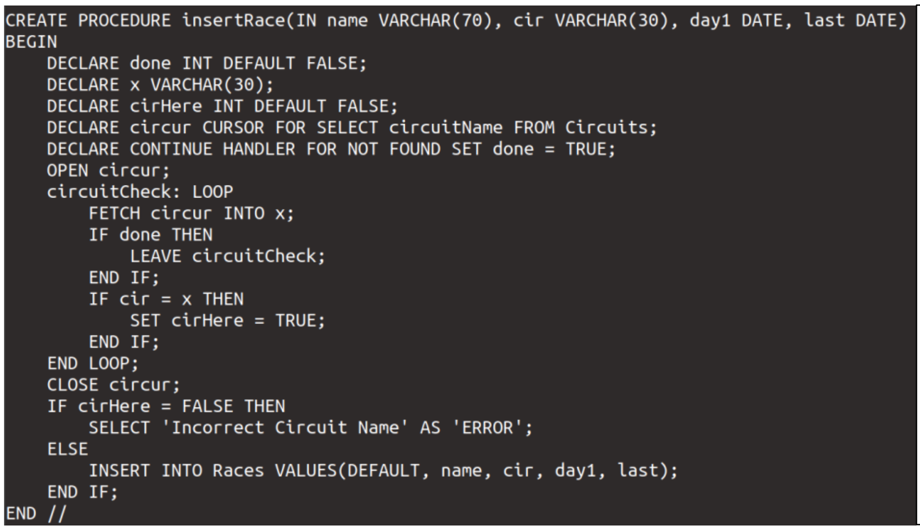
4. insertCircuit()



```
CREATE PROCEDURE insertCircuit(IN name VARCHAR(30), place VARCHAR(15), tyre CHAR(2))
BEGIN
    DECLARE tyreHere INT DEFAULT FALSE;
    CALL checkTyre(tyre, tyreHere);
    IF tyreHere = FALSE THEN
        SELECT 'Invalid Tyre Compound' AS 'ERROR';
    ELSEIF tyre = 'C4' THEN
        SELECT 'Please Enter The Hardest Tyre Compound That Will Be Used On This Circuit' AS 'ERROR';
    ELSEIF tyre = 'C5' THEN
        SELECT 'Please Enter The Hardest Tyre Compound That Will Be Used On This Circuit' AS 'ERROR';
    ELSE
        INSERT INTO Circuits VALUES(name, place, tyre);
    END IF;
END //
```

Procedure used to insert a circuit into the database. This also makes use of the previous checkTyre() stored procedure, to satisfy the foreign key constraint.

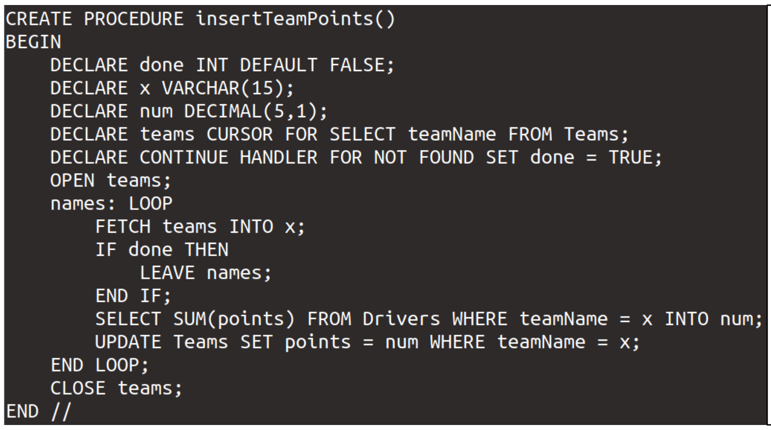
5. insertRace()



```
CREATE PROCEDURE insertRace(IN name VARCHAR(70), cir VARCHAR(30), day1 DATE, last DATE)
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE x VARCHAR(30);
    DECLARE cirHere INT DEFAULT FALSE;
    DECLARE circur CURSOR FOR SELECT circuitName FROM Circuits;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
    OPEN circur;
    circuitCheck: LOOP
        FETCH circur INTO x;
        IF done THEN
            LEAVE circuitCheck;
        END IF;
        IF cir = x THEN
            SET cirHere = TRUE;
        END IF;
    END LOOP;
    CLOSE circur;
    IF cirHere = FALSE THEN
        SELECT 'Incorrect Circuit Name' AS 'ERROR';
    ELSE
        INSERT INTO Races VALUES(DEFAULT, name, cir, day1, last);
    END IF;
END //
```

Procedure made to insert a race into the database. Use of cursor to check if the circuit name entered is valid. Does not take raceID as a parameter, as that is an auto-increment variable, utilizing the incrementRace trigger.

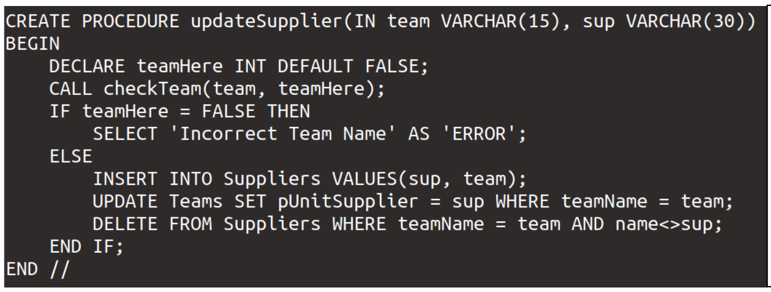
6. insertTeamPoints()



```
CREATE PROCEDURE insertTeamPoints()
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE x VARCHAR(15);
    DECLARE num DECIMAL(5,1);
    DECLARE teams CURSOR FOR SELECT teamName FROM Teams;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
    OPEN teams;
    names: LOOP
        FETCH teams INTO x;
        IF done THEN
            LEAVE names;
        END IF;
        SELECT SUM(points) FROM Drivers WHERE teamName = x INTO num;
        UPDATE Teams SET points = num WHERE teamName = x;
    END LOOP;
    CLOSE teams;
END //
```

Procedure used to insert or update current points the teams have, depending on the current points of their representative drivers. Utilizes a cursor to loop through each teamName and uses them to add points from their respective drivers.

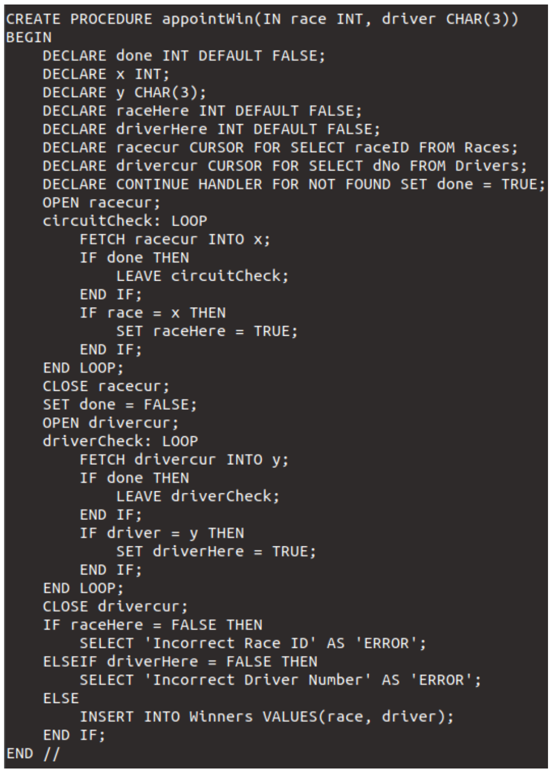
7. updateSupplier()



```
CREATE PROCEDURE updateSupplier(IN team VARCHAR(15), sup VARCHAR(30))
BEGIN
    DECLARE teamHere INT DEFAULT FALSE;
    CALL checkTeam(team, teamHere);
    IF teamHere = FALSE THEN
        SELECT 'Incorrect Team Name' AS 'ERROR';
    ELSE
        INSERT INTO Suppliers VALUES(sup, team);
        UPDATE Teams SET pUnitSupplier = sup WHERE teamName = team;
        DELETE FROM Suppliers WHERE teamName = team AND name<>sup;
    END IF;
END //
```

Procedure used to update the supplier of a team. Uses the checkTeam() stored procedure to perform the foreign key constraint check.

8. appointWin()



```
CREATE PROCEDURE appointWin(IN race INT, driver CHAR(3))
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE x INT;
    DECLARE y CHAR(3);
    DECLARE raceHere INT DEFAULT FALSE;
    DECLARE driverHere INT DEFAULT FALSE;
    DECLARE racecur CURSOR FOR SELECT raceID FROM Races;
    DECLARE drivercur CURSOR FOR SELECT dNo FROM Drivers;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
    OPEN racecur;
    circuitCheck: LOOP
        FETCH racecur INTO x;
        IF done THEN
            LEAVE circuitCheck;
        END IF;
        IF race = x THEN
            SET raceHere = TRUE;
        END IF;
    END LOOP;
    CLOSE racecur;
    SET done = FALSE;
    OPEN drivercur;
    driverCheck: LOOP
        FETCH drivercur INTO y;
        IF done THEN
            LEAVE driverCheck;
        END IF;
        IF driver = y THEN
            SET driverHere = TRUE;
        END IF;
    END LOOP;
    CLOSE drivercur;
    IF raceHere = FALSE THEN
        SELECT 'Incorrect Race ID' AS 'ERROR';
    ELSEIF driverHere = FALSE THEN
        SELECT 'Incorrect Driver Number' AS 'ERROR';
    ELSE
        INSERT INTO Winners VALUES(race, driver);
    END IF;
END //
```

Stored procedure used to insert a tuple into the Winners table. Because both the attributes in the table are foreign keys, there need to be two separate foreign key constraint checks; one for the raceID, entered as the first IN parameter, and the other for the dNo, entered as the second IN parameter. Utilizing two cursors for each check, and one continue handler that resets after the first loop is complete.

Triggers:

1. incrementRace

```
CREATE TRIGGER incrementRace
BEFORE INSERT ON Races FOR EACH ROW
BEGIN
    DECLARE maxID INT;
    SELECT MAX(raceID)+1 FROM Races INTO maxID;
    SET NEW.raceID = maxID;
END //
```

Trigger used to automatically increment the raceID BEFORE INSERT of a new tuple into the Races table.

2. updateTeamPoints

```
CREATE TRIGGER updateTeamPoints
AFTER UPDATE ON Drivers FOR EACH ROW
CALL insertTeamPoints();
```

Trigger used to update the team points after a driver's points have been modified. Calls the insertTeamPoints() stored procedure.

Views:

1. driverStand

```
CREATE VIEW driverStand AS
SELECT dNo, CONCAT(fName, ' ', lName) AS 'Full Name', teamName, points
FROM Drivers ORDER BY points DESC;
```

This VIEW is the perfect way to keep track of the current standings of the drivers, as it virtually updates the points when they are modified in the actual table. View ordered by the points to make it a leaderboard.

2. constructorStand

```
CREATE VIEW constructorStand AS
SELECT * FROM Teams ORDER BY points DESC;
```

Simple and effective View. Used to find the current standings of the constructors, which is the most defining part for the constructors championship and is some of the most viewed data by users. Easy choice for a view.

Python Connectivity:

After the implementation of the Formula 1 database was complete, the connection with python was made, for ease of use and efficiency of time when trying to run the designed and implemented queries shown.

After the connector file is run, it will allow the users to run and display all the queries listed above, with the addition of some other SELECT queries and VIEWS, with control and without the need to execute a sql script file.

All the runnable features have been added as string variables individually and are executed using the .execute() function from a cursor object, made from the mysql connection object.

A menu system is made, which gives options to the user, so they can run their queries of choice. Each choice uses the .execute() function, enclosed in a switch case made from if, elif, and else conditions in a pythonic way. The results are then displayed using a for loop, printing all elements in the cursor.

Final Discussion:

Challenges:

There were numerous challenges even just going into the assignment.

Since the requirements were open-ended, it did give all the freedom to make what I wanted, but that came with the long brainstorming of what it was I actually wanted. There would be no time redo things already put into action, due to the tight schedule, so designing and creating things like the ER diagram in a way where all the future tasks would also be easy was difficult and took a significant amount of time. Simply put, starting the assignment was a challenge.

Future Improvements:

- Creating the database to have same functionality, for multiple seasons instead of just one.
- Providing the user with more options, directly from the python connector.
- Making more features to analyze the data present, to make predictions.
- Adding graphical elements to the database, for example pictures of the cars, to give users a better idea of them.